Sequential decisions under uncertainty
Policy iteration

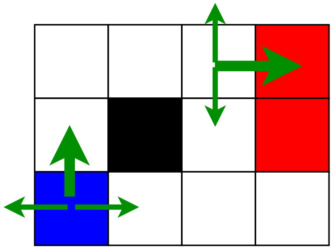
Tomáš Svoboda & Matej Hoffmann

Department of Cybernetics, Vision for Robotics and Autonomous Systems,
Center for Machine Perception (CMP)

May 28, 2018

Unreliable actions in observable grid world

- Walls block movement – agent/robot stays in place.
- Actions do not always go as planned.
- Agent receives rewards each time step:
  - Small "living" reward/penalty.
  - Big rewards/penalties at the end.
- Goal: maximize sum of (discounted rewards)



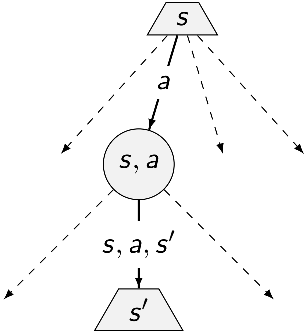
MDPs recap

Markov decision processes (MDPs):

- Set of states S
- Set of actions A
- Transitions $P(s'|s,a)$ or $T(s,a,s')$
- Rewards $R(s)$; and discount γ

MDP quantities:

- Policy $\pi(s) : S \rightarrow A$
- Utility – sum of (discounted) rewards.
- Values – expected future utility from a state (max-node)
- Q-Values – expected future utility from a q-state (chance-node)



Q-values – like values but given that I have commited to do action a from state s.

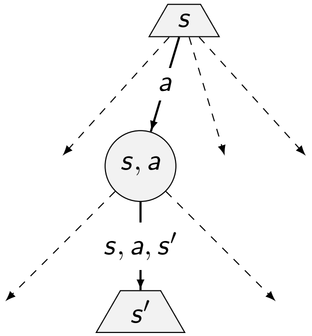
MDPs recap

Markov decision processes (MDPs):

- Set of states S
- Set of actions A
- Transitions $P(s'|s,a)$ or $T(s,a,s')$
- Rewards $R(s)$; and discount γ

MDP quantities:

- Policy $\pi(s) : S \to A$
- Utility – sum of (discounted) rewards.
- Values – expected future utility from a state (max-node)
- Q-Values – expected future utility from a q-state (chance-node)



Q-values – like values but given that I have commited to do action a from state s.

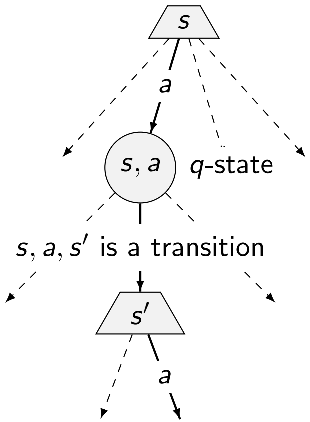
Optimal quantities

- The optimal policy: $\pi^*(s)$ – optimal action from state s
- Expected utility of a policy.

$$U^\pi = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

Best policy π^* maximizes above.

- The value of a state s: $V^*(s)$ – expected utility starting in s and acting optimally.
- The value of a q-state (s, a): $Q^*(s, a)$ – expected utility having taken a from state s and acting optimally thereafter.



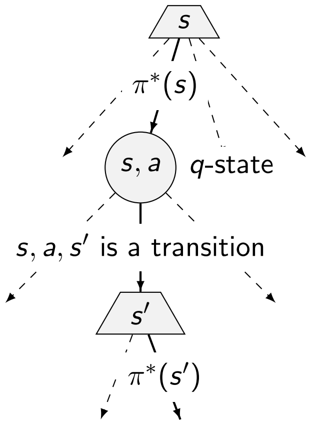
Optimal quantities

- The optimal policy: $\pi^*(s)$ – optimal action from state s
- Expected utility of a policy.

$$U^\pi = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

Best policy π^* maximizes above.

- The value of a state s: $V^*(s)$ – expected utility starting in s and acting optimally.
- The value of a q-state (s, a): $Q^*(s, a)$ – expected utility having taken a from state s and acting optimally thereafter.



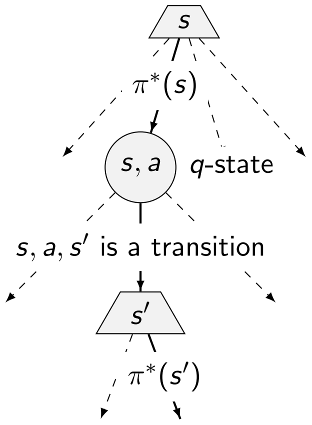
Optimal quantities

- The optimal policy: $\pi^*(s)$ – optimal action from state s
- Expected utility of a policy.

$$U^\pi = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

Best policy π^* maximizes above.

- The value of a state s: $V^*(s)$ – expected utility starting in s and acting optimally.
- The value of a q-state (s, a): $Q^*(s, a)$ - expected utility having taken a from state s and acting optimally thereafter.

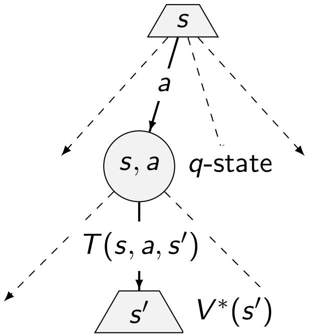


V^* and Q^*

The value of a q-state (s, a):
$Q^*(s,a) = \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma\, V^*(s')]$

The value of a state s:

$$V^*(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V^*(s')$$
$$= \max_a Q^*(s,a)$$

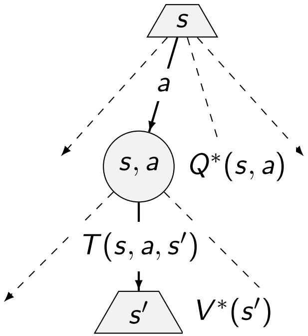


V^* and Q^*

The value of a q-state (s, a):
$Q^*(s,a) = \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma\, V^*(s')]$

The value of a state s:

$$V^*(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V^*(s')$$
$$= \max_a Q^*(s,a)$$

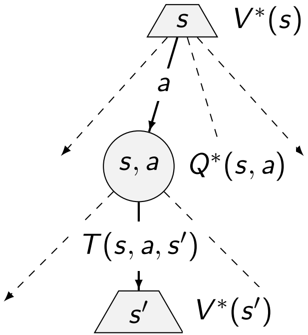


V^* and Q^*

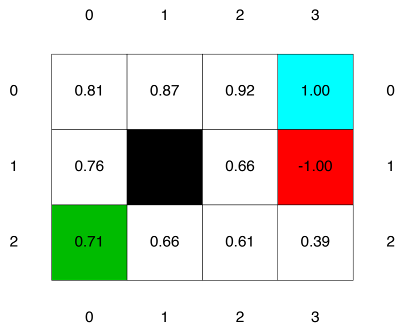
The value of a q-state (s, a):
$Q^*(s,a) = \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma\, V^*(s')]$

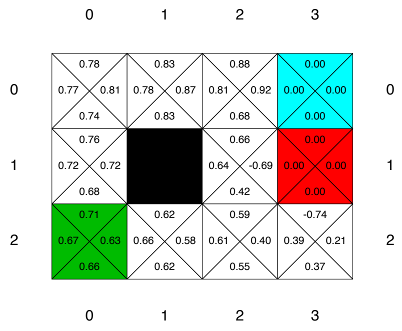
The value of a state s:

$$\begin{aligned} V^*(s) &= R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V^*(s') \\ &= \max_a Q^*(s,a) \end{aligned}$$



Maze: V^* vs. Q^*





$$
\begin{aligned}
Q^*(s,a) &= R(s) + \gamma \sum_{s'} T(s,a,s') V^*(s') \\
V^*(s) &= \max_a Q^*(s,a)
\end{aligned}
$$

This is the $R = -0.04$ for nonterminal states maze (AIMA Fig. 17.3).
Note that the Value of a state takes into account a number of things:

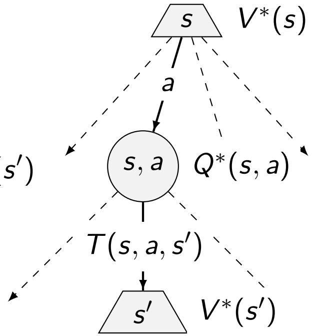
- the policy – which action will chosen in s
- the fact that the goal may be far away and
  - there will be a number of living penalties incured before reaching it
  - the final reward will be discounted
- the transition probabilities

Q-values - useful for choosing the best action – getting the policy.

Value iteration

- Bellman equations characterize the optimal values

$$V^*(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V^*(s')$$



- Value iteration computes them:

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V_k(s')$$

Value iteration is a fixed point solution method.

Bellman equations:

1. Take correct first action (1 ply of Expectimax)
2. Keep being optimal (recursion $V^*(s')$)

But note that this is a system of equations – no need to actually recurse

...

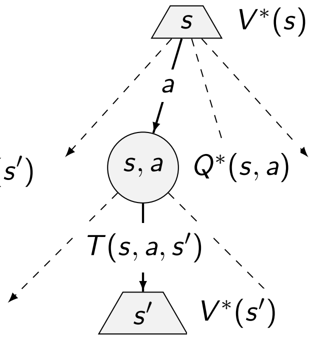
Value iteration

- Bellman equations characterize the optimal values

$$V^*(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V^*(s')$$

- Value iteration computes them:

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V_k(s')$$



Value iteration is a fixed point solution method.

Bellman equations:

1. Take correct first action (1 ply of Expectimax)
2. Keep being optimal (recursion $V^*(s')$)

But note that this is a system of equations – no need to actually recurse

. . .

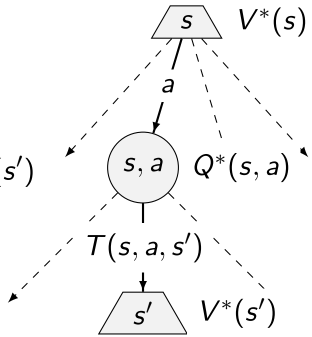
Value iteration

- Bellman equations characterize the optimal values

$$V^*(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V^*(s')$$

- Value iteration computes them:

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V_k(s')$$



Value iteration is a fixed point solution method.

Bellman equations:

1. Take correct first action (1 ply of Expectimax)
2. Keep being optimal (recursion $V^*(s')$)

But note that this is a system of equations – no need to actually recurse

...

Convergence

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s, a, s') V_k(s')$$

- Think about special cases: deterministic world, $\gamma = 0$, $\gamma = 1$.
- For all s, $V_k(s)$ and $V_{k+1}(s)$ can be seen as expectimax search trees of depth k and $k + 1$

From Values to Policy

Value iteration

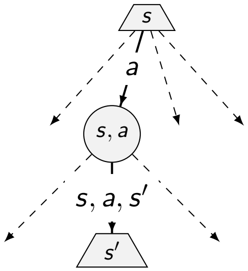
- Start with arbitrary $V_0(s)$
- Compute Bellman update (one ply of expectimax from each state)

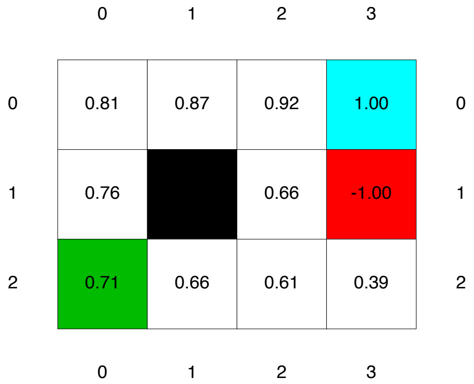
$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) V_k(s')$$

- Repeat until convergence

Value iteration was supposed to give me the policy. But where is the policy?

Policy extraction - computing actions from Values

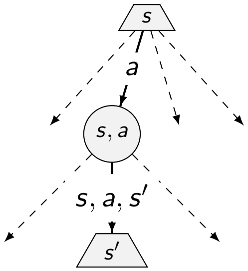




- Assume we have $V^*(s)$
- What is the optimal action?
- We need a one-step expectimax:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} T(s, a, s') V^*(s')$$

Policy extraction - computing actions from Values



	0	1	2	3	
0	0.81	0.87	0.92	1.00	0
1	0.76		0.66	-1.00	1
2	0.71	0.66	0.61	0.39	2
	0	1	2	3	

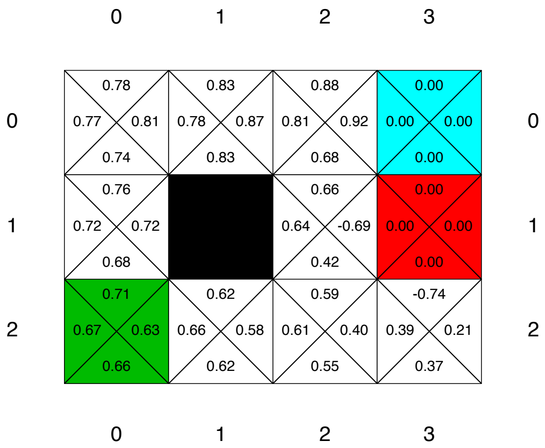
- Assume we have $V^*(s)$
- What is the optimal action?
- We need a one-step expectimax:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} T(s, a, s') V^*(s')$$

Policy extraction - computing actions from Q-Values

- Assume we have $Q^*(s, a)$
- What is the optimal action?
- Just take the (arg) max:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s, a)$$

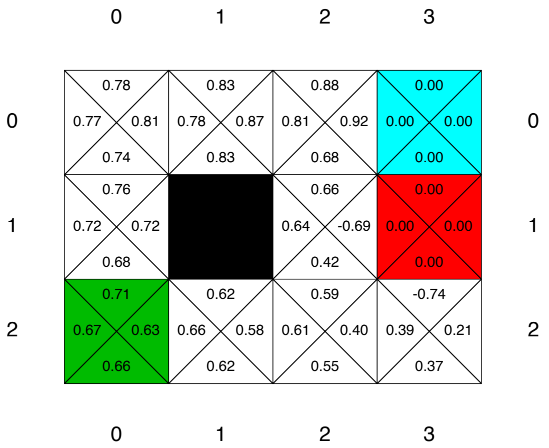


Actions are easier to extract from q-values.

Policy extraction - computing actions from Q-Values

- Assume we have $Q^*(s, a)$
- What is the optimal action?
- Just take the (arg) max:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s, a)$$



Actions are easier to extract from q-values.

What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V_k(s')$$

- What is complexity of one iteration - over all S states?
- Does the "max" change often?
- When the does the policy converge?
- Can we compute the policy directly?

Complexity: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.
Policy often converges long before the values.

What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s, a, s') V_k(s')$$

- What is complexity of one iteration - over all S states?
- Does the "max" change often?
- When the does the policy converge?
- Can we compute the policy directly?

Complexity: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.

Policy often converges long before the values.

What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V_k(s')$$

- What is complexity of one iteration - over all S states?
- Does the "max" change often?
- When the does the policy converge?
- Can we compute the policy directly?

Complexity: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.
Policy often converges long before the values.

What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V_k(s')$$

- What is complexity of one iteration - over all S states?
- Does the "max" change often?
- When the does the policy converge?
- Can we compute the policy directly?

Complexity: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.

Policy often converges long before the values.

What is wrong with the Value iteration?

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s,a,s') V_k(s')$$

- What is complexity of one iteration - over all S states?
- Does the "max" change often?
- When the does the policy converge?
- Can we compute the policy directly?

Complexity: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.

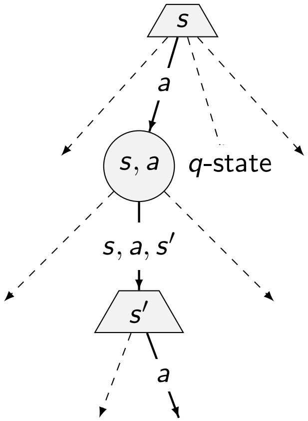
Policy often converges long before the values.

Policy evaluation

- Assume $\pi(s)$ given.
- How to evaluate (compare)?

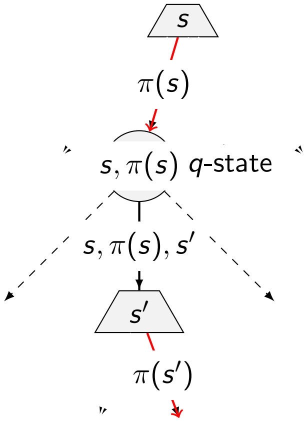
Remember last week's quizz?

Fixed policy, do what π says



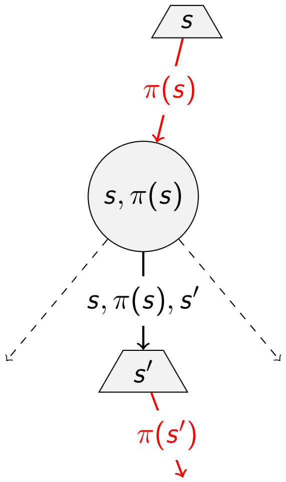
- Expectimax trees "max" over all actions
  ...
- Fixed π for each state $\rightarrow$ no "max" operator!

Fixed policy, do what π says



- Expectimax trees "max" over all actions
  ...
- Fixed π for each state $\rightarrow$ no "max" operator!

State values under a fixed policy



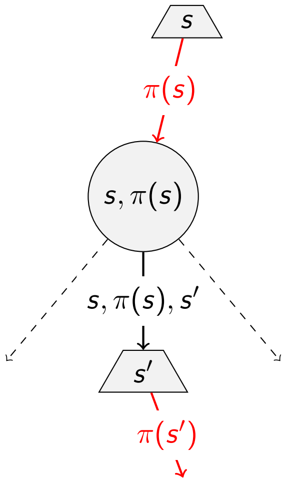
- Expectimax trees "max" over all actions

  . . .

- Fixed π for each state $\rightarrow$ no "max" operator!

$V^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^\pi(s')$

State values under a fixed policy



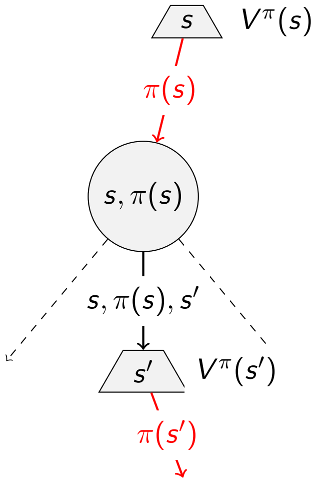
- Expectimax trees "max" over all actions

  . . .

- Fixed π for each state $\rightarrow$ no "max" operator!

$V^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s')$

State values under a fixed policy



- Expectimax trees "max" over all actions
  ...
- Fixed π for each state $\rightarrow$ no "max" operator!

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s')$$

How to compute $V^\pi(s)$?

$$V^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^\pi(s')$$

Policy iteration

- Start with a random policy.
- Step 1: Evaluate it.
- Step 2: Improve it.
- Repeat steps until policy converges.

Policy iteration

- Start with a random policy.
- Step 1: Evaluate it.
- Step 2: Improve it.
- Repeat steps until policy converges.

Policy iteration

- Start with a random policy.
- Step 1: Evaluate it.
- Step 2: Improve it.
- Repeat steps until policy converges.

Policy iteration

- Start with a random policy.
- Step 1: Evaluate it.
- Step 2: Improve it.
- Repeat steps until policy converges.

Policy iteration

- Policy evaluation. Solve equations or iterate until convergence.

$$V_{k+1}^{\pi_i}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V_k^{\pi_i}(s')$$

- Policy improvement. Look-ahead. Policy extraction from fixed values.

$$\pi_{i+1}(s) = \arg\max_{a \in A(s)} \sum_{s'} T(s, a, s') V^{\pi_i}(s')$$

Policy iteration algorithm

function POLICY-ITERATION(env) **returns:** policy π
  input: env - MDP problem
  $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
  $V(s) \leftarrow 0$ in all states
  repeat ▷ iterate values until no change
    $V \leftarrow$ POLICY-EVALUATION(π, V, env)
    unchanged $\leftarrow$ True
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ **then**
        $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
        unchanged $\leftarrow$ False
      end if
    end for
  until unchanged
end function

Policy iteration algorithm

function POLICY-ITERATION(env) **returns:** policy π
  input: env - MDP problem
  $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
  $V(s) \leftarrow 0$ in all states
  repeat ▷ iterate values until no change
    $V \leftarrow$ POLICY-EVALUATION(π, V, env)
    unchanged $\leftarrow$ True
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ **then**
        $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
        unchanged $\leftarrow$ False
      end if
    end for
  until unchanged
end function

Policy iteration algorithm

function POLICY-ITERATION(env) **returns:** policy π
    input: env - MDP problem
    $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
    $V(s) \leftarrow 0$ in all states
    repeat ▷ iterate values until no change
        $V \leftarrow$ POLICY-EVALUATION(π, V, env)
        unchanged $\leftarrow$ True
        for each state s **in** S **do**
            if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ **then**
                $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
                unchanged $\leftarrow$ False
            end if
        end for
    until unchanged
end function

Policy iteration algorithm

function POLICY-ITERATION(env) **returns:** policy π
  input: env - MDP problem
  $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
  $V(s) \leftarrow 0$ in all states
  repeat ▷ iterate values until no change
    $V \leftarrow$ POLICY-EVALUATION(π, V, env)
    unchanged $\leftarrow$ True
    for each state s **in** S **do**
      if $\max\limits_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ **then**
        $\pi(s) \leftarrow \arg\max\limits_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
        unchanged $\leftarrow$ False
      end if
    end for
  until unchanged
end function

Policy iteration algorithm

function POLICY-ITERATION(env) **returns:** policy π
  input: env - MDP problem
  $\pi(s) \leftarrow$ random $a \in A(s)$ in all states
  $V(s) \leftarrow 0$ in all states
  repeat ▷ iterate values until no change
    $V \leftarrow$ POLICY-EVALUATION(π, V, env)
    unchanged $\leftarrow$ True
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s') > \sum_{s'} P(s'|s,\pi(s))V(s')$ **then**
        $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|a,s)V(s')$
        unchanged $\leftarrow$ False
      end if
    end for
  until unchanged
end function

Policy vs. Value iteration

- Value iteration.
  - Iteration updates values and policy. Although policy implicitly – extracted from values
  - No track of policy.
- Policy iteration.
  - Update utilities is fast – only one action per state.
  - New policy from values (slower)
  - New policy is better or done.
- Both methods belong to Dynamic programming realm.

Complexity:
Value iteration: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.
Policy often converges long before the values.
Policy iteration: $O(S^3)$ (after AIMA, pg. 657)
The Bellman equations are *linear* because the max operator is gone.
For $\sharp S$ states, we have $\sharp S$ equations, which can be solved exactly in time $O(S^3)$ using standard linear algebra methods.
For small state spaces - ok.
For large state spaces - may be prohibitive $\rightarrow$ *modified policy iteration* with only a certain number of simplified Bellman update.

Policy vs. Value iteration

- Value iteration.
  - Iteration updates values and policy. Although policy implicitly – extracted from values
  - No track of policy.
- Policy iteration.
  - Update utilities is fast – only one action per state.
  - New policy from values (slower)
  - New policy is better or done.
- Both methods belong to Dynamic programming realm.

Complexity:
Value iteration: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.
Policy often converges long before the values.
Policy iteration: $O(S^3)$ (after AIMA, pg. 657)
The Bellman equations are *linear* because the max operator is gone.
For $\sharp S$ states, we have $\sharp S$ equations, which can be solved exactly in time $O(S^3)$ using standard linear algebra methods.
For small state spaces - ok.
For large state spaces - may be prohibitive $\rightarrow$ *modified policy iteration* with only a certain number of simplified Bellman update.

Policy vs. Value iteration

- Value iteration.
  - Iteration updates values and policy. Although policy implicitly – extracted from values
  - No track of policy.
- Policy iteration.
  - Update utilities is fast – only one action per state.
  - New policy from values (slower)
  - New policy is better or done.
- Both methods belong to Dynamic programming realm.

Complexity:
Value iteration: $O(S^2 * A)$
For every state (LHS), there can be up to $\sharp S$ also on RHS – if every other state was reachable from the current state.
In addition, all actions from every state need to be considered.
$Max(A)$ **does not** change often.
Policy often converges long before the values.
Policy iteration: $O(S^3)$ (after AIMA, pg. 657)
The Bellman equations are *linear* because the max operator is gone.
For $\sharp S$ states, we have $\sharp S$ equations, which can be solved exactly in time $O(S^3)$ using standard linear algebra methods.
For small state spaces - ok.
For large state spaces - may be prohibitive $\rightarrow$ *modified policy iteration* with only a certain number of simplified Bellman update.

References

Further reading: Chapter 17 of [1] however, policy iteration is quite compact there. More detailed discussion can be found in chapter Dynamic programming in [2] with slightly different notation, though. This lecture has been also greatly inspired by the 9th lecture of CS 188 at http://ai.berkeley.edu as it convincingly motivates policy search and offers an alternative convergence proof of the value iteration method.

[1] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 3rd edition, 2010.
http://aima.cs.berkeley.edu/.

[2] Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning; an Introduction.
MIT Press, 2nd edition, 2018.
http://www.incompleteideas.net/book/bookdraft2018jan1.pdf.

Bandits